كلية الهندسة بشبرا
FACULTY OF ENGINEERING AT SHOUBRA

Communications Engineering Department
Shoubra Faculty of Engineering
Benha University

Traffic light controller

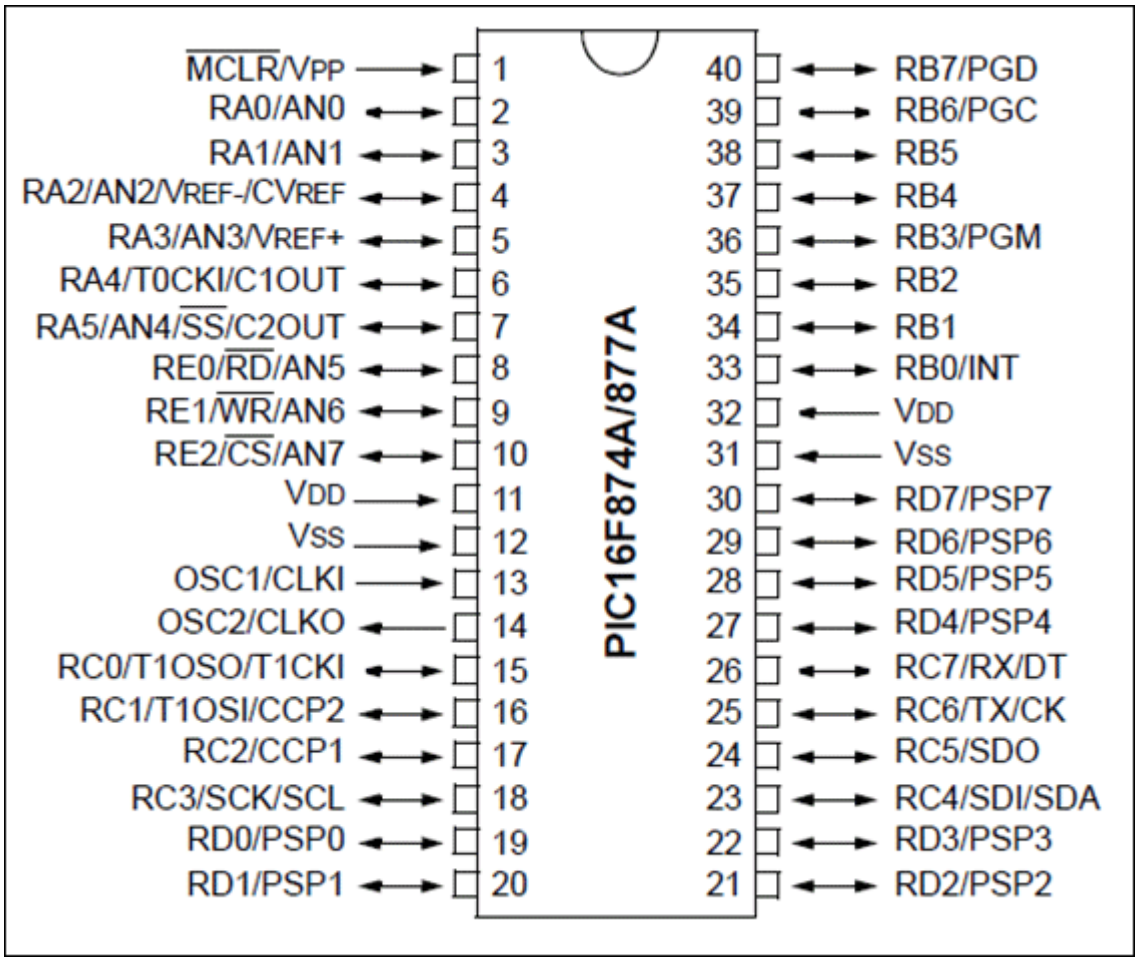
Written by: Abdallah Amin Ebrahim
SEC: 2 B.N: 35

2nd Year Communications Engineering

1. Task1

1.a) Describe all the pins of PIC16f877A. After that, your colleagues would have enough information once they need to interface the PIC16f877A with other hardware.

- There are 40 pins in PIC16f877A



- **PIN 1: MCLR:** Master clear pin resets the microcontroller and is active low, meaning that it should constantly be given a voltage of 5V and if 0 V are given then the controller is reset. Resetting the controller will bring it back to the first line of the program that has been burned into the IC.
- **PIN 2-7: PORT A:** PORTA consists of 6 pins, from pin 2 to pin 7, all of these are input/output pins.
    - **PIN 2: RA0/AN0:** Pin 2 is the first pin of this port. This pin can also be used as an analog pin AN0. It is built in **analog to digital converter**.
    - **PIN 3: RA1/AN1:** This can be the analog input 1.
    - **PIN 4: RA2/AN2/Vref-: It** can also act as the analog input2. Or negative analog reference voltage can be given to it.

- **PIN 5: RA3/AN3/Vref+:** It can act as the analog input 3. Or can act as the analog positive reference voltage.
- **PIN 6: RA4/T0CKI:** To timer0 this pin can act as the clock input pin.
- **PIN 7: RA5/SS/AN4:**This can be the analog input 4.

- **PIN 8-10: PORT E:** PORTE starts from pin 8 to pin 10 and this is also a input output port
  - **PIN 8: RE0/RD/AN5:** It can be the analog input 5.
  - **PIN 9: RE1/WR/AN6:** It can be the analog input 6.
  - **PIN 10: RE2/CS/A7:** It can be the analog input 7.
- **PIN 11 and 32: VDD:** These two pins are the positive supply for the input/output and logic pins. Both of them should be connected to 5V.
- **PIN 12 and 31: VSS:** These pins are the ground reference for input/output and logic pins. They should be connected to 0 potential.
- **PIN 13: OSC1/CLKIN:** This is the oscillator input or the external clock input pin.
- **PIN 14: OSC2/CLKOUT:** This is the oscillator output pin.

A crystal resonator is connected between pin 13 and 14 to provide external clock to the microcontroller.

- **PIN 15-26: PORT C:** PORTC consists of 8 pins. It is also a input output port. It starts from pin 15 to pin 26.
  - **PIN 15: RC0/T1OCO/T1CKI:** pin 15 is the first. It can be the clock input of timer 1 or the oscillator output of timer 2.
  - **PIN 16: RC1/T1OSI/CCP2:** It can be the oscillator input of timer 1 or the capture 2 input/compare 2 output/ PWM 2 output.
  - **PIN 17: RC2/CCP1:** It can be the capture 1 input/ compare 1 output/ PWM 1 output.
  - **PIN 18: RC3/SCK/SCL:** It can be the output for SPI or I2C modes and can be the input/output for synchronous serial clock.
  - **PIN 23: RC4/SDI/SDA:** It can be the SPI data in pin. Or in I2C mode it can be data input/output pin.
  - **PIN 24: RC5/SDO:** It can be the data out of SPI in the SPI mode.
  - **PIN 25: RC6/TX/CK:** It can be the synchronous clock or USART Asynchronous transmit pin.
  - **PIN 26: RC7/RX/DT:** It can be the synchronous data pin or the USART receive pin.
- **PIN 19-30:** All of these pins belong to PORTD which is a input and output port. When the microprocessor bus is to be interfaced, it can act as the parallel slave port.

- **PIN 33-40: PORT B:** All these pins belong to PORTB. Out of which RB0 can be used as the external interrupt pin and RB6 and RB7 can be used as in-circuit debugger pins.

1. b) Explain to your colleagues the functions of the main blocks in PIC16f877A : ALU, Status and Control, Program Counter, Flash Program Memory, Instruction Register, Instruction Decoder.

- **Arithmetic Logic Unit (ALU):** The ALU is responsible for performing arithmetic (addition, subtraction) and logic (AND, OR, XOR) operations. It is a critical component in executing instructions that involve mathematical calculations or decision-making processes.
- **Status and Control**: These registers hold flags that indicate the current status of the ALU operations and the microcontroller as a whole. The most important status flags include:
  - **Carry Flag (C):** Indicates if an arithmetic operation resulted in a carry out of the most significant bit.
  - **Zero Flag (Z):** Indicates if an operation resulted in a zero value.
  - **Digital Carry Flag (DC):** Used in BCD (Binary-Coded Decimal) operations.
  - **Negative Flag (N):** Indicates if the result of an operation is negative.
- **Program Counter (PC)**: The Program Counter is a special register that holds the address of the next instruction to be executed. It increments automatically after each instruction, guiding the microcontroller through the sequence of instructions stored in memory.
- **Flash Program Memory**: This is the non-volatile memory where the program code (instructions) is stored. Since it is Flash memory, it retains its content even when the power is turned off, and it can be rewritten during programming.
- **Instruction Register**: The Instruction Register temporarily holds the current instruction fetched from Flash Program Memory before it is executed. This register feeds the instruction to the Instruction Decoder.
- **Instruction Decoder**: The Instruction Decoder interprets the instruction stored in the Instruction Register and determines what operation needs to be performed. It generates the necessary control signals to execute the instruction using the ALU, registers, and other microcontroller components.

1. c) Examine the reasons why a led, which is connected to RA4 for flashing prepose not working probably.

RA4 is an open-drain output, meaning it can only sink current and cannot source it. To properly use an LED with RA4, the LED's anode should be connected to a positive supply, and the cathode should be connected to RA4. This way, RA4 can pull the cathode low to turn the LED on, but it cannot drive the LED high by itself.

1. d) ATMega328P is also an 8-bit but AVR microcontroller. Evaluate the characteristics of ATMega328P versus PIC16f877A, by comparing the memory size, the power consumption, pin count… of those two MCUs. Give 2 examples of embedded systems where ATMega328P is a better choice than PIC16f877A.

Feature	ATMega328P	PIC16f877A
Memory Size		
Flash Memory	32 KB	14 KB
EEPROM	1 KB	256 Bytes
Power Consumption		
Operating Voltage	1.8V to 5.5V	2.0V to 5.5V
Active Mode	0.2 mA at 1 MHz, 1.8V	1.6 mA at 4 MHz, 5V
Power-down Mode	0.1 µA at 1.8V	1 µA at 2V
Pin Count		
DIP Package	28 pins	40 pins
TQFP Package	32 pins	44 pins
Clock Speed	20 MHz	20 MHz
Architecture		
Type	AVR (8-bit RISC)	PIC (8-bit)
Instruction Set	RISC, single-cycle instructions	CISC, multiple-cycle instructions
Peripherals		
ADC Channels	6 channels, 10-bit	8 channels, 10-bit
Timers/Counters	3 (1 x 16-bit, 2 x 8-bit)	3 (1 x 16-bit, 2 x 8-bit)
PWM Channels	6	2
USART	1	1
SPI	1	1
I2C	1	1
Cost	Generally higher	Typically lower

Two examples where the ATMega328P would be a better choice than the PIC16f877A:

1. Arduino Projects

The ATMega328P would be a better choice than the PIC16f877A due to:

- **Community and Support**: The ATMega328P is widely used in Arduino boards, which have extensive community support, libraries, and tutorials. This makes it easier to develop, troubleshoot, and expand Arduino projects.
- **Development Tools**: The ease of use with the Arduino IDE and the availability of numerous libraries simplifies the development process, especially for beginners.

2. 3D Printer Controller

The ATMega328P would be a better choice than the PIC16f877A due to:

- **Wide Ecosystem**: The ATMega328P is commonly used in popular 3D printer controller boards (like the RAMPS board), which have a vast array of community support, firmware options, and tutorials.
- **Processing Speed**: The microcontroller's ability to efficiently handle real-time tasks is crucial for precise control of 3D printing operations.

2. Task2

2.1 Introduction

The project aims to design and implement a traffic light controller using the PIC16f877A microcontroller. The traffic light controller is intended to manage the flow of vehicles at an intersection. The system will feature two operational modes: manual and automatic. In automatic mode, the lights will switch between red, yellow, and green at pre-set intervals for two streets, West and South. In manual mode, a user can manually control the lights via switches, with a mandatory 3-second yellow light interval.

2.2 PIC16f877A Overview

The PIC16f877A microcontroller is an 8-bit device from Microchip Technology, widely known for its versatility and robust performance in various embedded system applications. This microcontroller features 368 bytes of RAM, 256 bytes of EEPROM, and 8K words of flash program memory, providing sufficient memory resources for small to medium-scale projects.

Operating at a maximum clock frequency of 20 MHz, the PIC16f877A includes a range of peripherals that make it suitable for tasks requiring precise control and timing. These peripherals include multiple timers, analog-to-digital converters (ADCs), and communication interfaces like UART, SPI, and I²C, which allow for seamless integration with other devices.

A significant advantage of the PIC16f877A is its extensive I/O capabilities. The microcontroller offers 33 I/O pins across five ports, providing the flexibility needed to interface with various external components, such as LEDs, switches, and 7-segment displays. Additionally, the PIC16f877A supports multiple interrupt sources, enabling efficient handling of real-time events, which is crucial for applications like a traffic light controller.

2.3 System Design(circuit)

The traffic light controller circuit is designed to manage traffic flow at an intersection with two streets, West and South. The system utilizes the PIC16f877A microcontroller to control the traffic lights and display countdown timers using 7-segment displays. The key components of the circuit include:

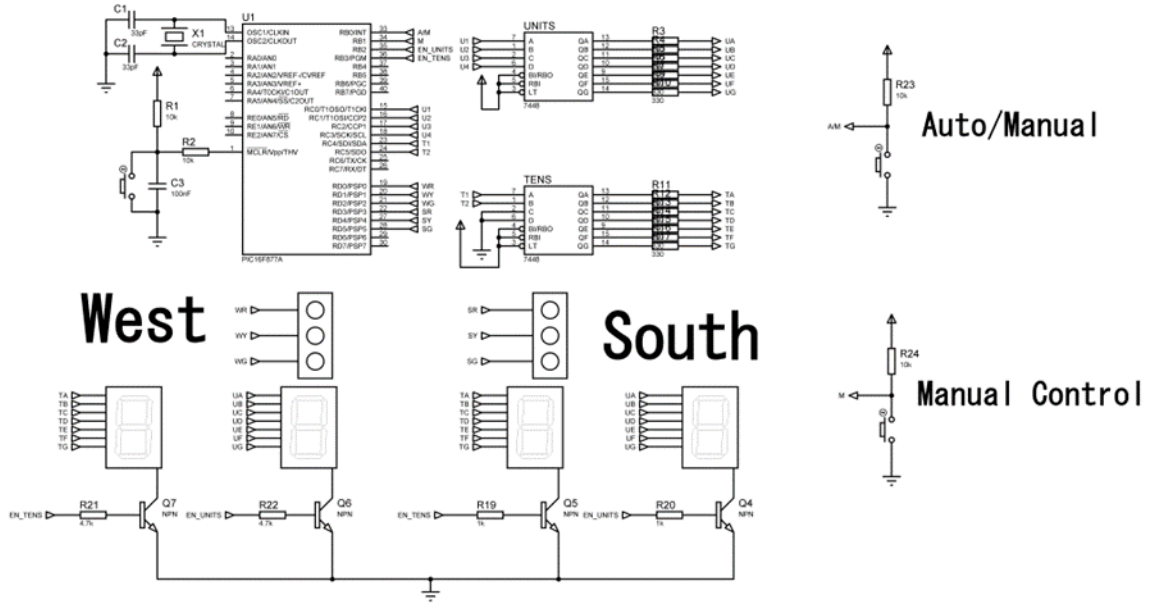
- **PIC16f877A Microcontroller**: Acts as the central control unit, managing the operation of traffic lights and handling user inputs.
- **LEDs**: Represent the traffic lights for each street. Each street has three LEDs (red, yellow, and green) controlled by specific output pins on the microcontroller.

- **7-Segment Displays**: Used to show the countdown timers for each street. The displays are driven by a 7447 BCD to 7-segment decoder IC to reduce the number of microcontroller pins required.
- **BJTs (Transistors)**: Used as switches to enable or disable the 7-segment displays.
- **Push-Button Switches**: Allow the user to switch between manual and automatic modes and to toggle between the two streets in manual mode.

Automatic Mode: In this mode, the microcontroller automatically cycles through the traffic light sequence. The West street is displayed with red for 15 seconds, yellow for 3 seconds, and green for 20 seconds. Following this, the South street shows red for 23 seconds, yellow for 3 seconds, and green for 12 seconds. The system enforces these timing intervals to ensure smooth and safe traffic flow.

Manual Mode: In manual mode, the user can manually control the traffic lights using the push-button switches. One switch toggles between manual and automatic modes, while the other switch allows the user to manually switch the active street. Even in manual mode, the system ensures a 3-second yellow light.

Circuit Diagram



2.4 Software Design(code)

```
 #define autoManualSwitch PORTB.RB0
#define manualSwitch PORTB.RB1
#define enableUnits PORTB.RB2
#define enableTens PORTB.RB3
#define westRedLed PORTD.RD0
#define westYellowLed PORTD.RD1
#define westGreenLed PORTD.RD2
#define southRedLed PORTD.RD3
#define southYellowLed PORTD.RD4
#define southGreenLed PORTD.RD5

char current_mode = 0;
char i;

void interrupt() {
   if (INTCON.INTF) {
      Delay_ms(30); // Debounce delay
      INTCON.INTF = 0; // Clear the interrupt flag
      current_mode = (current_mode == 0 ? 1 : 0); // Toggle mode
   }
}

void setCounter(char seconds) {
   char units = seconds % 10;
   char tens = seconds / 10;

   if (seconds == 0) {
      enableUnits = 0;
      enableTens = 0;
   } else {
      enableUnits = (seconds < 10) ? 1 : 1;
      enableTens = (seconds >= 10) ? 1 : 0;
   }
   PORTC = (units & 0x0F) | ((tens & 0x0F) << 4);
}

void main() {
   // Initialize ports
   TRISB = 0x03; // RB0, RB1 as inputs (switches), others as outputs
   TRISC = 0x00; // PORTC as output (7-segment display)
   TRISD = 0x00; // PORTD as output (LEDs)
   PORTC = 0x00; // Clear PORTC
   PORTD = 0x00; // Clear PORTD
   enableUnits = 1;
   enableTens = 1;

   // Configure interrupts
   INTCON.GIE = 1; // Enable global interrupts
```

```
INTCON.INTE = 1; // Enable INT external interrupt
INTCON.INTF = 0; // Clear the interrupt flag
OPTION_REG.INTEDG = 1; // Interrupt on rising edge

while (1) {
   if (!current_mode) { // Automatic mode
      // Reset all LEDs
      southGreenLed = 0;
      southRedLed = 0;
      southYellowLed = 0;
      westGreenLed = 0;
      westYellowLed = 0;
      westRedLed = 1; // West street red light

      // West street green/yellow timing
      for (i = 15; i > 0 && !current_mode; i--) {
         southGreenLed = (i > 3 ? 1 : 0);
         southYellowLed = (i <= 3 ? 1 : 0);
         setCounter(i);
         Delay_ms(1000); // Wait 1 second
      }

      westRedLed = 0;
      southYellowLed = 0;
      southRedLed = 1; // South street red light

      // South street green/yellow timing
      for (i = 23; i > 0 && !current_mode; i--) {
         westGreenLed = (i > 3 ? 1 : 0);
         westYellowLed = (i <= 3 ? 1 : 0);
         setCounter(i);
         Delay_ms(1000); // Wait 1 second
      }
      southRedLed = 0;
      westYellowLed = 0;
   } else { // Manual mode
      if (westRedLed) { // If West street is in red light
         for (i = 3; i > 0 && current_mode; i--) {
            southYellowLed = 1;
            southGreenLed = 0;
            setCounter(i);
            Delay_ms(1000); // Wait 1 second
         }
         while (current_mode && manualSwitch == 1) {
            westRedLed = 0;
            westYellowLed = 0;
            westGreenLed = 1;
            southRedLed = 1;
            southYellowLed = 0;
            southGreenLed = 0;
```

```
                setCounter(0);
                Delay_ms(50); // Short delay to reflect manual switch state
            }
        } else { // If West street is in yellow light
            for (i = 3; i > 0 && current_mode; i--) {
                westYellowLed = 1;
                westGreenLed = 0;
                setCounter(i);
                Delay_ms(1000); // Wait 1 second
            }
            while (current_mode && manualSwitch == 1) {
                westRedLed = 1;
                westYellowLed = 0;
                westGreenLed = 0;
                southRedLed = 0;
                southYellowLed = 0;
                southGreenLed = 1;
                setCounter(0);
                Delay_ms(50); // Short delay to reflect manual switch state
            }
        }
    }
  }
}
```